A fuzzy galois lattices approach to requirements elicitation for cloud services

Todoran Koitz, Irina ; Glinz, Martin

Abstract: The cloud paradigm has become increasingly attractive throughout the recent years due to its both technical and economic foreseen impact. Therefore, researchers' and practitioners' attention has been drawn to enhancing the technological characteristics of cloud services, such as performance, scalability or security. However, the topic of identifying and understanding cloud consumers' real needs has largely been ignored. Existing requirements elicitation methods are not appropriate for the cloud computing domain, where consumers are highly heterogeneous and geographically distributed, have frequent change requests and expect services to be delivered at a fast pace. In this paper, we introduce a new approach to requirements elicitation for cloud services, which utilizes consumers' advanced search queries for services to infer requirements that can lead to new cloud solutions. For this, starting from the queries, we build fuzzy Galois lattices that can be used by public cloud providers to analyze market needs and trends, as well as optimum solutions for satisfying the largest populations possible with a minimum set of features implemented. This new approach complements the existing requirements elicitation techniques in that it is a dedicated cloud method which operates with data that already exists, without entailing the active participation of consumers and requirements specialists.

DOI: https://doi.org/10.1109/TSC.2015.2466538

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2466538, IEEE Transactions on Services Computing

A Fuzzy Galois Lattices Approach to Requirements Elicitation for Cloud Services

Irina Todoran Koitz, *Student Member, IEEE,* and Martin Glinz, *Member, IEEE Computer Society*

Abstract—The cloud paradigm has become increasingly attractive throughout the recent years due to its both technical and economic foreseen impact. Therefore, researchers' and practitioners' attention has been drawn to enhancing the technological characteristics of cloud services, such as performance, scalability or security. However, the topic of identifying and understanding cloud consumers' real needs has largely been ignored. Existing requirements elicitation methods are not appropriate for the cloud computing domain, where consumers are highly heterogeneous and geographically distributed, have frequent change requests and expect services to be delivered at a fast pace. In this paper, we introduce a new approach to requirements elicitation for cloud services, which utilizes consumers' advanced search queries for services to infer requirements that can lead to new cloud solutions. For this, starting from the queries, we build fuzzy Galois lattices that can be used by public cloud providers to analyze market needs and trends, as well as optimum solutions for satisfying the largest populations possible with a minimum set of features implemented. This new approach complements the existing requirements elicitation techniques in that it is a dedicated cloud method which operates with data that already exists, without entailing the active participation of consumers and requirements specialists.

Index Terms—cloud services, data analysis, requirements elicitation, Galois lattice.

1 Introduction

Cloud computing is largely seen as a successful and promising paradigm due to its capability to efficiently adapt to business changes by scaling software or hardware resources in a flexible way. Therefore, it has received great interest from both research and industry throughout the recent years, and has so far managed to maintain its promise to deliver both technical and economic benefits [1].

As a result, the number of public cloud services available is growing continuously and it is expected that this growth will continue in the future [2]. While this can be seen as an advantage for cloud consumers who have a wide variety of offers to choose from, this phenomenon can also lead to a paradox of choice [3], where users do not know what services best match their needs. To solve this problem, researchers [4] and industry practitioners recognize that there is a need to develop search engines or platforms dedicated to aggregating and displaying cloud service offerings from various providers. These would act as marketplaces [5], [6] exposing advanced search capabilities that allow (potential) cloud consumers to input and refine their needs according to various criteria. Then, a matching algorithm would identify what existing public cloud services match the features requested. An example in this direction is the Intel Cloud Finder [7], which matches IT requirements to existing cloud services from those providers that signed up on the Intel platform and published their offering.

Another consequence of the rapid growth in popularity of cloud services is the emergence of numerous related research areas, ranging from intercloud architecture models to cloud performance and virtualization. In this context, the focus has been strongly directed towards building better services from a technological perspective, whereas the human aspect has been largely ignored. While there are examples of research conducted in the area of cloud adoption [8], [9], the issue of identifying and satisfying cloud consumers' real needs has not been thoroughly addressed. Moreover, the existing requirements acquisition or elicitation techniques are not suitable for the cloud domain, and dedicated requirements elicitation methods for the cloud are lacking [10]. In this paper, we introduce a new approach for solving an existing requirements engineering (RE) problem: the absence of dedicated requirements elicitation techniques for cloud services. We concentrate on identifying (potential) cloud consumers' needs by modeling and analyzing the data collected from consumers' advanced search queries on cloud service marketplaces.

In the following subsections, we firstly explain what requirements elicitation is and why it is important. Then, we clarify the cloud challenges that hinder the usage of existing requirements elicitation methods and finally outline the main contributions of this paper.

1.1 Requirements Elicitation

Requirements elicitation is typically seen as the first step in the requirements engineering process [11]. According to Sommerville and Kotonya, requirements elicitation refers to activities undertaken to discover the requirements of a system to be built or a problem to be solved [12]. Additionally, Van Lamsweerde also includes the identification of stakeholders in the requirements elicitation stage [13]. Generally, requirements elicitation refers to seeking, gathering and consolidating requirements, and is regarded as an indispensable step towards building successful solutions.

Nuseibeh et al. highlight that requirements are not somewhere, waiting to be collected, but elicitation techniques are

• *I. Todoran Koitz and M. Glinz are with the Department of Informatics, University of Zurich, Switzerland.*
E-mails: {koitz, glinz}@ifi.uzh.ch

Manuscript received January 29, 2015; revised June 19, 2015

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2466538, IEEE Transactions on Services Computing

necessary to investigate and understand users' needs [14]. For instance, traditional methods (e.g., questionnaires, interviews, analysis of existing documentation), group elicitation methods (e.g., brainstorming, focus groups, RAD/JAD workshops), prototyping, model-driven techniques (e.g., scenarios, KAOS, i*), cognitive methods (e.g., protocol analysis, laddering, card sorting) and contextual techniques were developed to enable requirements elicitation, and have been used successfully in traditional settings for decades.

Similarly, in the cloud domain, consumers' requirements have to be identified in order to know what characteristics future cloud services should exhibit such that they satisfy consumers' needs, and to avoid failure-proneness. However, the cloud paradigm poses a few challenges that do not allow using the existing requirements elicitation methods, as explained in the following subsection.

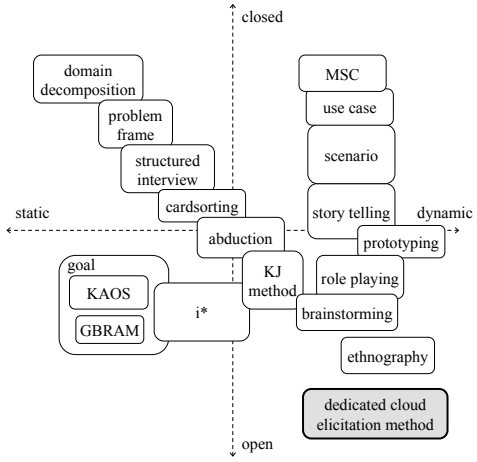

Figure 1. Requirements elicitation techniques, adapted after [16].

1.2 Cloud Challenges

Whereas existing requirements elicitation methods have proven useful for determining stakeholders' needs in traditional contexts, i.e. where stakeholders are easy to identify and physically reachable, most of these techniques are heavily challenged by the particular features of the cloud. For instance, given that cloud services can be easily sold and customized online, consumers are generally *geographically distributed*, often worldwide. This leads to a *lack of local markets*, i.e. cloud providers do not always have a deep understanding of the international markets they sell to. This is in contrast to the traditional delivery model, where contracts are made with local physical suppliers that then sell software or hardware solutions, intermediating this way the expansion of the business to a local, known market.

Moreover, cloud consumers can be highly *numerous and heterogeneous*, with diverse profiles and backgrounds, and exhibiting thoroughly different requirements that cannot be easily satisfied on an individual basis. In such settings, existing elicitation methods cannot be applied, since stakeholders cannot be identified, such that requirements specialists can interact with them directly [15].

Another challenge is represented by the *frequent change requests* coming from cloud consumers, especially businesses, and their *volatile requirements*. On the one hand, such cloud consumers are largely modern businesses that need their requests to be met fast. On the other hand, to face the competition, providers have to first know, ideally predict, and then satisfy these requests efficiently, such that they do not lose their clients. In this context, they cannot afford to apply elicitation methods that require long waiting times for gathering requirements, or long processing and analyzing times, and this is where most of the existing techniques fail.

Last but not least, the cloud is still *young* compared to the traditional delivery model. Therefore, dedicated methods for addressing consumers' needs have not been developed so far, to address the challenges identified.

Tsumaki and Tamai [16] categorize the existing requirements elicitation methods according to two criteria. Firstly, depending on how requirements acquisition is conducted, requirements can be collected and sorted either in a static or dynamic way. Secondly, depending on the properties of the target space to be analyzed, the space can be either closed or open. Using this categorization, due to the fast and dynamic pace of the cloud, service providers should elicit requirements in a *dynamic*, ideally continuous fashion. Since consumers' needs may change rapidly and this can often be unpredictable, the space is *open*. As it can be observed in Figure 1, methods such as brainstorming, role playing or ethnography could seemingly fit these characteristics. However, these are the type of methods that necessarily require the physical and simultaneous presence of stakeholders in the same geographical space, which is incompatible with the cloud paradigm. Therefore, according to the existing related work and based on our previous research [17], it is evident that there is a need for dedicated cloud elicitation techniques that support cloud companies in understanding and providing their consumers with services that truly meet their needs. Such a dedicated method would belong in the framework proposed by Tsumaki and Tamai in the bottom right corner (dynamic elicitation process and open space), as shown in Figure 1, and should accommodate the following requirements [17]:

R1: Fit for wide and heterogeneous audiences;
R2: Take less time than traditional elicitation methods;
R3: Make automated elicitation possible;
R4: Be applied remotely;
R5: Be able to handle volatile requirements.

1.3 Main Contributions

This work is an extension of an existing paper published in the 2014 IEEE 10th World Congress on Services [18]. There, we introduced the preliminary idea of building fuzzy Galois lattices to support cloud providers in the requirements elicitation activity. In this extension, we have three new contributions:

1. We enhance the preliminary algorithm introduced in [18], by improving its performance through a pre-processing technique that calculates frequencies of search queries. This is included as a functionality in the StakeCloud Tool we developed.

2. We add a similarity classifier that allows the flexible clustering of similar queries; this leads to reducing the modeling space, thus improving the scalability. Similarly, this functionality is included in the prototype.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2466538, IEEE Transactions on Services Computing

3. We conduct a series of experimental evaluations to verify our requirements elicitation approach, and explain how it meets the five requirements above.

The remainder of the paper is organized as follows. Section 2 describes our approach, including the idea, definitions for the terms used and algorithms developed. Section 3 presents the evaluation of the solution and the interpretation of the results. In Section 4, we give an overview of related work, and Section 5 concludes the paper.

2 Approach

2.1 Definitions

In this subsection, we introduce the terms utilized in describing our approach, as well as the corresponding mathematical definitions.

Definition 1. A *partially ordered set (or poset)* is a set taken together with a partial order on it. Formally, a partially ordered set is defined as an ordered pair $A = (X, \leq)$, where X is called the ground set of A and $\leq$ is the partial order of A.

Definition 2. For any subset A' of a poset A, the members of the families $lb(A') = \{a \in A : \forall a' \in A' : a \leq a'\}$ and $ub(A') = \{a \in A : \forall a' \in A' : a' \leq a\}$ are called the *lower* and *upper bounds* of A' in A, respectively.

Definition 3. The members of the families $inf(A') = max(lb(A'))$ and $sup(A') = min(ub(A'))$ are called *infima* and *suprema* of A' in A, respectively.

In other words, the *supremum* of A' is the smallest element of A that is greater than or equal to each element of A'. It is unique and it may or may not belong to A'. The *infimum* of A' is defined analogously.

Definition 4. A poset A is called a *Galois lattice* iff for any subset A' of A, there exist a least upper bound $sup \in A$ (the *supremum* of A') and a least lower bound $inf \in A$ (the *infimum* of A').

Galois connections have their roots in Galois theory [19], and refer to correspondences between two posets. According to Erné et al. [20], they are defined as follows:

Definition 5. Considering the posets $\mathcal{P} = \langle P, \leq \rangle$ and $\mathcal{D} = \langle Q, \geq \rangle$, if $P \overset{\pi_*}{\to} Q$ and $Q \overset{\pi^*}{\to} P$ are functions such that for all $p \in P$ and all $q \in Q$, $p \leq q\pi^*$ iff $p\pi^* \sqsubseteq q$, then the quadruple $\pi = \langle \mathcal{P}, \pi_*, \pi^*, \mathcal{D} \rangle$ is called a *Galois connection*.

Definition 6. A *binary relation* $R(X, Y)$ is a set of ordered pairs $(x, y), x \in X, y \in Y$. For any given elements $p \in X$ and $q \in Y$, the pair (p, q) is either an element of $R(X, Y)$ or it is not.

In a Galois concept lattice, the elements can generally take binary values. When we model cloud service queries, some features can be easily represented using only binary values, e.g., the service provides mobile support (1) or not (0). However, numerous features are better represented on ratio scales, e.g., for data storage cloud services, the values can be between 1 GB and 20 TB; in this case, binary values and therefore binary relations would be difficult to use. Consequently, we use the extension of the Galois lattice theory to fuzzy binary relations [21], such that features can not only be represented on a nominal scale (taking the binary values 0 or 1), but also on a ratio scale.

Definition 7. *Fuzzy binary relations* $\tilde{R}$ allow a *degree of membership* in a relation: the degree of membership of (p, q) in $\tilde{R}$ may be any real number from the range $[0, 1]$.

Therefore, a fuzzy set S_i includes a degree of membership for each of its elements, taking a value in the range [0,1]. A set with the membership degrees restricted to the values 0 and 1 (crisp set) is a particular type of a fuzzy set, so it is formally correct to mix the features on a nominal scale with those on a ratio scale in the same representation.

Definition 8. Each concept in a Galois hierarchy that represents the set of objects sharing the same values for a certain set of properties is called a *formal concept*.

In contrast to general formal concept analysis theory (FCA) [22], Galois connections take into consideration the relations between fuzzy concepts represented on ratio scales. For example, 0.2 and 0.5 are only two distinct values in FCA, whereas 0.2 and 0.5 are two values which can be ordered in Galois connections theory, e.g., 0.2<0.5. This leads to the notion of fuzzy Galois lattices [21] the nodes of which represent fuzzy concepts, which in turn are constructed from a fuzzy binary relation.

2.2 Idea

With the emergence of cloud service marketplaces acting as intermediators between consumers and providers, large amounts of data are generated. (Potential) consumers use such platforms to search for services that match their criteria, thus inputting their needs and preferences as advanced search queries. Such log data are currently used by recommender systems [23] to suggest existing services with similar features, but this is generally the highest extent to which these data are exploited. A significant majority of the companies we interviewed in one of our previous studies [17] mentioned that they logged consumers' search data and tried to analyze it, but the large dimensions usually hindered the understanding. In most cases, their analysis was reduced to identifying which service features appeared most commonly in the advanced search queries. Therefore, we are facing a big data problem: the large datasets cannot be processed and visualized easily, which leads to losing the potential of such data.

Our idea is to utilize the logged advanced search queries for cloud services to find requirements and combinations of features that can eventually lead to developing new cloud services and new classes of cloud solutions. Our approach follows the process illustrated by the UML activity diagram in Figure 2. The process contains ten activities, as follows.

A1: Collect data. The input necessary for running our approach is represented by advanced search queries data collected through a cloud services marketplace. In this context, an advanced search query is a query that allows users to specify desired values for a set of predefined features.

A2: Check for duplicates. Once the data has been collected, our StakeCloud Tool that implements the approach checks the input dataset for duplicates, i.e. identical queries.

A3: Count frequency for each unique query. In case no duplicate queries are found, each query has the frequency set

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2466538, IEEE Transactions on Services Computing

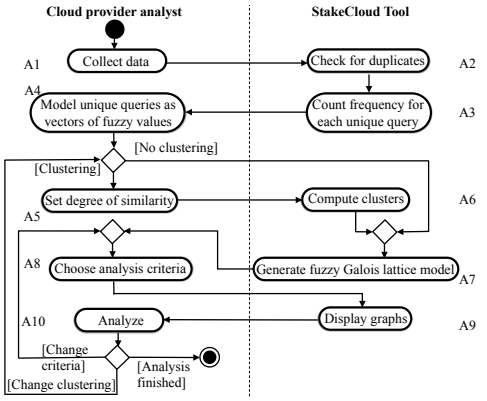

Figure 2. UML activity diagram of the process of our approach [24].

to 1. If duplicate queries are identified, the tool keeps only one instance of each unique query and counts its frequency, i.e. how many times it appears in the given dataset.

A4: Model unique queries as vectors of fuzzy values. Naturally, advanced search queries contain heterogeneous data. For instance, the values associated with a query input for a public cloud data storage service can be: 300GB storage capacity, 30 days file recovery, 98% reliability, and the service should allow AES encryption for backup. Using monotonic modeling functions, such values are transformed into fuzzy values, taking values in the range [0,1]. For example, the feature "reliability" can be modeled as a monotonic function as follows, transforming a value of 95% into the fuzzy value 0.5. Here, the range $(90, 99\%]$ is considered the most important and is modeled accordingly, since most services do not have reliability values under 90% or above 99%.

$$f(x) = \begin{cases} 0, x \leq 90\% \\ 0.1 * (x - 90), x \in (90, 99\%] \\ 1, x > 99\% \end{cases}$$

For features represented on a nominal scale, such as "AES encryption", $f(x)$ can take the value of 1 if the feature is available, else 0. The rest of features work similarly. Evidently, there are numerous ways in which $f(x)$ can be defined; the choice only has to ensure that it is a monotonic transformation which leads to values in the range [0,1], and maintains a ratio scale for fuzzy values. Table 1 shows a list of ten different features of a cloud data storage service, along with possible modeling functions. This way, an advanced query for a cloud service can be defined as a vector of fuzzy values $S_i = \{f_{ij} : j = 1, n\}$, where f represents the features of the cloud service, and n is the number of features assigned to the generic type of service S in the search platform. According to Figure 2, the activity of defining the modeling functions is performed by the cloud provider. However, our tool is enhanced with a set of pre-defined monotonic functions for common cloud service features, which can be used by the cloud provider to (semi-)automate this task. Moreover, the tool has a checking mechanism that ensures the functions defined by the provider cover the range [0,1] monotonously. The current tool does not support modeling exclusion queries, but we plan to integrate this in our next release. The undesired features can be marked in our dataset with a flag and then propagated in the lattices.

A5: Set degree of similarity. The cloud provider representative can bundle or cluster the input queries by selecting the degree of similarity on a scale from 0 to 100%. The degree of similarity is an integer in the range [0,100] and a similarity of 100% means that all the input queries are considered to be 100% similar, thus leading to one single cluster that includes all of them. Conversely, a similarity of 0% means that all the input queries are distinct, thus leading to a number of clusters equal to the number of queries, each cluster containing one single query. Provided that the degree of similarity is higher than 0, the bundling action will group similar queries and define a representative vector of fuzzy values for each such bundle (more details on this in Section 2.3.3). The bundling is calculated based on Euclidean distances between the vectors of features composing the input queries and the supremum of all the queries. Therefore, a vector is considered similar to another vector to a degree of, e.g., 20% when, among all the other vectors, it is within the closest 20% distance. When the input datasets are large, this activity may be necessary to enable an easier visualization of the output and achieve improved performance. For this reason, we allow the user to set the degree of similarity already before generating the lattice. However, the user can also directly generate the lattice without bundling.

Table 1
Service features (N = nominal scale: {0,1}, R = ratio scale: [0,1])

No.	Feature	Scale	Modeling function f(x)
f_1	Private user	N	0: N/A, 1: available
f_2	Business user	N	0: N/A, 1: available
f_3	Storage	R	$10^{-3}x, x < 10^3\ GB$; $1, x \geq 10^3\ GB$
f_4	Mobile support	N	0: N/A, 1: available
f_5	File recovery	R	$10^{-2}x, x < 10^2 days$; $1, x \geq 10^2 days$
f_6	Reliability	R	$0, x \leq 90\%; 1, x > 99\%$; $0.1 * (x - 90), x \in (90, 99\%]$
f_7	AES encryption	N	0: N/A, 1: available
f_8	SSL encryption	N	0: N/A, 1: available
f_9	Max size/file	R	$0, x < 0.1\ GB; 1, x \geq 10\ GB$ $0.1 * x, x \in [0.1, 10)\ GB$
f_{10}	Uptime	R	$0, x \leq 90\%; 1, x > 99\%$; $0.1 * (x - 90), x \in (90, 99\%]$

A6: Compute bundles. Provided that the user selected a degree of similarity different from the default value (0%), the tool computes the corresponding bundles and the representatives (as vectors of fuzzy values) for each. The algorithm used for this step is detailed in Section 2.3.3.

A7: Generate fuzzy Galois lattice model. Based on the input dataset modeled as vectors of fuzzy values, and/or the computed bundles, the tool generates the corresponding fuzzy Galois lattice (for an example, see Figure 7). Lattices are represented graphically as acyclic directed graphs having exactly one source node (with no incoming edges) and one sink node (with no outgoing edges). In our applied case of fuzzy Galois lattices, the nodes on the first level in the hierarchy correspond to the vectors of features given by cloud consumers (e.g., *(1), (2)*). As explained in Section 2.1, the topmost element is the supremum or upper bound for

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2466538, IEEE Transactions on Services Computing

all the lattice elements. This is the only lattice node that satisfies all input queries fully. However, implementing such a service in practice is most often either very expensive or impossible. Therefore, we recommend that cloud providers should analyze the infima options, which are represented by all the other nodes of the lattice, from the second hierarchy level down (e.g., *(1,4), (2,7), (2,4,7), (1,2,3,4)*). Infima nodes are service offerings that satisfy the input queries, but only to a limited extent. This way, providers can make compromises to satisfy large populations of consumers with a minimum set of requirements implemented, to reach an optimum solution. In this respect, it is important to note that all the infima elements represent newly generated classes of services, which can be candidates for implementation. Therefore, our approach goes beyond simple statistics, since the infima elements are new combinations of feature values, which cannot be easily inferred by counting frequencies in the initial datasets or using standard statistical methods. Moreover, if the lattice generated is split in sub-lattices for a more thorough analysis, the same properties related to infima and suprema elements are maintained for the sub-models: any sub-lattice has at least a supremum and an infimum node, respectively, so in any such subset of nodes there will be at least a service candidate that fully satisfies the nodes in the first hierarchy level and at least a node that satisfies these to a limited extent, which can be calculated by our method. The algorithm employed for building these models is described in Section 2.3.2.

A8: Choose analysis criteria. Once the lattice has been drawn (including bundles or not), the cloud provider user of our approach can choose the preferred criteria for performing data analysis. These criteria are selected and implemented in such a way that they support the service provider in his/her understanding and reasoning process, to decide what types of services should be supplied to satisfy consumers' needs. For instance, (s)he can choose to compare the satisfaction level for individual features for a set of classes of services (graph nodes) (s)he selected in the lattice, or (s)he can study to what extent particular queries can be satisfied by new classes of services generated by our method. These analysis criteria are detailed in Section 3.5.2.

A9: Display graphs. Based on the criteria chosen, the corresponding graphs are displayed. The graphical representation can consist of points or functions in a Cartesian coordinate system (e.g., as shown in Figure 7).

A10: Analyze. The cloud company representative can use the lattice model and graphical representations from A9 to perform a thorough analysis of consumers' queries. Moreover, (s)he can change the analysis criteria at any time and generate new graphs, or change the degree of similarity used for bundling.

2.3 Algorithms

The activities that compose the process of our approach introduced in Section 2.2 are implemented by the following algorithms.

2.3.1 Frequency Counter

In order to support activities A2 and A3, we utilize Algorithm 1, shown in pseudocode below. This calculates the frequency for each query of the input file.

Algorithm 1 calculateFrequency(inputFile)

1: $d = dictionary(fuzzyVector)$
2: **foreach** $query\ q \in inputFile$ **do**
3: **if** $q \notin d.keys()$ **then**
4: $d[q] = 1$
5: **else**
6: $d[q] = d[q] + 1$
7: **end if**
8: **end for**

As an abstract data type, we use a dictionary (or associative array) composed of $(key, value)$ pairs. In our case, the values contained by the dictionary are vectors of fuzzy values ($fuzzyVector$), and each key appears only once. For every query q of the initial input file, we check whether it is part of the keys set or not. If it is found, we increment its frequency counter; if not, the frequency is set to 1. The output consists of a dataset made of exclusively unique queries and the corresponding computed frequencies. This algorithm ensures that the dataset to be processed further does not contain any duplicates, which is important for the overall performance and behaviour of the method. Algorithm 1 has $O(S)$ complexity, where S is the number of service queries.

2.3.2 Lattice Generator

The activity A7 is one of the core steps in our approach, since it deals with generating the fuzzy Galois lattice, the pre-requisite model for the data analysis. Algorithm 2 shows what operations are needed before the graph can be drawn.

Algorithm 2 generateLattice(fBR[S,F])

1: $C = C' = \emptyset$
2: $C_S^k = \binom{S}{k}$
3: $C = \cup_{k=1}^{S} \{C_S^k\}$
4: **foreach** $i \in C$ **do**
5: **foreach** $j \in i$ **do**
6: **foreach** $f \in [1..F]$ **do**
7: $infFeature[f] = min_{r=1}^{len(j)} j[r, f]$
8: $C' = C' \cup infFeature[f]$
9: **end for**
10: **end for**
11: **end for**
12: eliminateDuplicates(C')
13: drawGraph(C')

The input of Algorithm 2 is represented by a fuzzy binary relation (fBR), which is a matrix with two dimensions: S service queries that exist in the dataset with unique queries, and F features, which are the pre-defined features for the type of service analyzed, e.g., cloud data storage, as shown in Table 1. Firstly, all elements of the power set $\mathcal{P}(S)$ of the set S of search queries are generated as combinations (line 2). These are sets of sets of vectors with fuzzy values, where S queries are taken k at a time without repetition. For example, if $S = 5$ and $k = 3$, $C_5^3 = \{[s_1, s_2, s_3], [s_1, s_2, s_4], [s_1, s_2, s_5], [s_1, s_3, s_4], [s_1, s_3, s_5], [s_1, s_4, s_5], [s_2, s_3, s_4], [s_2, s_3, s_5], [s_2, s_4, s_5], [s_3, s_4, s_5]\}$, where $s_i, i = 1, S$ are unique service queries. Secondly, we append all these C_S^k calculated to C (line 3), which becomes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2466538, IEEE Transactions on Services Computing

a large set that includes all the possible combinations of service queries, based on the initial input file.

Thirdly, we calculate the fuzzy concepts (lines 4-11), yielding 2^S FCs. According to Galois connections theory, the FC belonging to a subset S' is calculated by taking the minima of all feature values of the queries contained in S' (line 7). In case C exposes special properties, these are considered at this stage. As a typical example, assume we detect a small distance between two rows of the fuzzy binary relation. Since this method is based on computing minimum values, detecting a search query which is the minimum of another will lead to reduction opportunities in the final lattice, i.e. some nodes do not have to be represented due to redundancy.

Naturally, after calculating all the FCs, C' may include duplicates. In order not to draw any classes of services more than once, we now eliminate the duplicates from C'. This way, we keep only one entry for each fuzzy vector generated when calculating the feature values (line 7).

The last step of Algorithm 2 consists of drawing the lattice (graph). When doing so, the algorithm takes into account the hierarchical properties of the lattice. For instance, our approach will draw the directed edges $FC_4 - FC_{4,5}$ and $FC_5 - FC_{4,5}$ from service queries s_4 and s_5 represented by FC_4 and FC_5, respectively, to the lattice node $FC_{4,5}$, which is a formal concept generated based on queries s_4 and s_5. The other edges are drawn in a similar fashion, e.g., $FC_{4,5} - FC_{2,4,5}$, $FC_{2,4,5} - FC_{1,2,4,5}$, such that the indices of the supremum node always represent a subset of the indices of the infimum node.

The operation for calculating the minima values in line 7 has a complexity of $O(\frac{2^S}{S})$ which, combined with the complexity of iterating over the combinations $O(2^S - 1)$ and features $O(F)$, leads to the overall complexity for Algorithm 2 of $O(\frac{F*2^{2S}}{S})$, therefore $O(2^S)$. The exponential character in the size of S is natural when computing Galois lattices.

2.3.3 Similarity Classifier

The third algorighm is used for implementing activity A6 (cf. Figure 2), where the tool computes clusters of similar queries based on a degree of similarity sim provided by the user in activity A5.

Algorithm 3 takes the degree of similarity sim and the set of unique queries ($uniqueQ$) as input. The latter is the output of activity A4, and is a bi-dimensional matrix with S queries and F service features. The algorithm returns the $clustersList$, which is a list containing all the clusters computed. Initially, this is initialized to the empty set. Our bundling algorithm uses Euclidean distances between vectors to find clusters, in a similar way to other clustering algorithms from data mining, such as k-means. Nevertheless, we could not have used an existing known implementation such as Lloyd's algorithm, since we cannot provide the number of expected clusters (the k value in k-means), but are interested in seeing the resulting clusters, regardless of their number. In this respect, our algorithm belongs to hierarchical clustering, avoiding the problem of first identifying the number of clusters generated. Moreover, it has a deterministic behavior, always producing the same results for the same input. Therefore, instead of selecting some random queries as anchors for the clusters, we use one single anchor: the supremum of the input dataset sup.

Algorithm 3 calculateSimilarity(sim,uniqueQ[S,F])

```
1:  clustersList = ∅
2:  if sim == 0 then
3:      return uniqueQ[S, F]
4:  else
5:      sup = supremum(uniqueQ)
6:      uniqueQ'[S', F'] = uniqueQ[S, F]
7:      foreach query q ∈ uniqueQ do
8:          dist(sup, q) = sqrt(Σ_{i=1}^{F} (sup_i − q_i)^2)
9:      end for
10:     while uniqueQ' ≠ ∅ do
11:         maxDist = max_{i=1,S',q_i∈uniqueQ'} (dist(sup, q_i))
12:         cluster = ∅
13:         breakingPoint = (sim * maxDist) / 100
14:         foreach query q ∈ uniqueQ' do
15:             if dist(sup, q) < breakingPoint then
16:                 cluster = cluster ∪ q
17:             end if
18:         end for
19:         uniqueQ' = uniqueQ' − {q|q ∈ cluster}
20:         clustersList = clustersList ∪ cluster
21:     end while
22:     return clustersList
23: end if
```

In case the degree of similarity sim given by the user is equal to 0, the algorithm returns the list of initial unique queries (lines 2-3) - the clusters list for $sim == 0$ coincides with the initial dataset when all the input queries are unique. Otherwise, if the sim value is greater than 0, we calculate the maximum Euclidean distance between the supremum sup and each query q in our set (lines 7-8), and the breaking point for the cluster (lines 11-13). The $breakingPoint$ defines the lower bound of the cluster, i.e. the least Euclidean distance to the supremum within which a search query must be in order to qualify for cluster membership. All the queries whose distances to the supremum are smaller than the $breakingPoint$ value for a given sim are considered similar for the sim value specified. Each element of the set ends up in a cluster, as long as the distance between it and the supremum is smaller than the breaking point (lines 14-18). Then, the dataset is updated (line 19) to include only those elements that did not become members of clusters or were left out as single unclustered elements, and the steps above are repeated until the dataset is empty. Finally, the clusters formed in each iteration are added to the $clustersList$ (line 20).

For calculating the representative of each cluster (a vector of fuzzy values), we use the centroid or geometric center concept. This is calculated as the arithmetic mean position of all the points in the n-dimensional space, where n is the number of features for the specified cloud service. We did not choose the medoid concept as the representative object for each cluster, as it is most commonly done in data mining, since medoids are always elements of the dataset, and this was not a requirement in our case. Here, it is more important to achieve a minimal dissimilarity between

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2466538, IEEE Transactions on Services Computing

the representative and all the elements of the cluster, such that the new lattice generated after the bundling activity illustrates the initial dataset well, without high information loss. If the queries included in the cluster have frequencies higher than 1, these are also taken into account, since we calculate a weighted centroid, i.e. each query is represented in the cluster proportionally to its frequency.

Given that we first test whether sim is 0, the best case complexity for Algorithm 3 is $O(1)$. Otherwise, since calculating $maxDist$ is done in $O(N')$, the overall complexity is $O(N+N'^2)$, where N is the number of elements of $uniqueQ$ and N' is the number of elements of $uniqueQ'$. The worst case complexity tends to $O(N^2)$, when N' tends to N.

Table 2
Product Managers' ideas on using the sample dataset

Activity	PM1	PM2	PM3	PM4	PM5
Analyze query frequency	x	x	x		x
Analyze query importance					x
Predict future requests		x		x	
Analyze queries for services similar to theirs			x	x	

3 Evaluation

3.1 Goal and Metrics

To evaluate our approach, we assess how it meets the five requirements introduced in Section 1.2 (R1-R5). These emerged from the related work in the field of dedicated cloud elicitation techniques and our previous study with 19 cloud provider companies [17]. Therefore, we assume that having an approach that meets these requirements would support cloud providers in understanding and satisfying their consumers' needs better. We define the goal for our evaluation as follows:

> Analyze *our dedicated requirements elicitation approach* for the purpose of *evaluation* with respect to *the extent to which it satisfies cloud providers' requirements for a new requirements elicitation method (R1-R5)* from the point of view of *cloud provider companies* in the context of *analyzing advanced search queries for cloud services to infer new requirements*.

Since R2 deals with time-efficiency, the main metric we use for meeting this requirement is the time. We calculate the time needed for generating the lattice nodes and models, as well as the time required for finding similar queries and generating clusters. As far as the automation is concerned (R3), we show what output can be automatically generated by our approach and explain to what extent the method introduced is more automatic than the existing requirements elicitation techniques. Furthermore, we use the standard deviation as a metric for heterogeneity (R1) of queries in a dataset. As far as remote application (R4) and volatile requirements (R5) are concerned, we describe how our approach can be applied and its output can be analyzed remotely, as well as how volatile requirements can be monitored and future predictions can be made.

3.2 Method

In this work, we focus on internal, rather than external evaluation. This means that we evaluate the performance of our approach and present how it meets the requirements identified (R1-R5), but do not conduct an external evaluation against other existing methods. The main reason for this is that such an evaluation is virtually impossible.

None of the existing requirements elicitation techniques mentioned in Section 1.1 uses advanced search queries datasets as input, such that we could compare the output of our method to the output of other similar methods. In this respect, the innovative nature of our approach is the cause for the lack of a benchmark or ground truth we could use for an external evaluation, e.g., using metrics such as precision, recall or fallout. Moreover, trying to analyze the datasets manually to build our own ground truth is impossible, given the large amounts of data.

3.3 Product Managers' Input

In our semi-structured interviews conducted with cloud providers between November 2012 and January 2013 on how they perform requirements elicitation [17], we discussed the possibility of utilizing consumers' search data for infering service requirements. At that time, while some companies were logging such data, none of the 19 interviewed cloud businesses was using them for requirements elicitation. In January 2015, we contacted again five of the product managers (PM) we had previously interviewed, from companies located in three different European countries, and asked them if this situation had changed meanwhile. All five explained that although the idea of using log data analysis for the purpose of getting to know their (potential) consumers better is a recurrent topic, no concrete initiatives have been taken in this regard so far. They confirmed that the requirements elicitation approaches they were using at the time we performed our interviews are still in use now.

Moreover, we now gave the five product managers a sample dataset of 500 advanced search queries, and asked them how they would use it to help their companies in their decision making process regarding the launch of a new cloud service. The most frequent responses recorded are displayed in Table 2. Analyzing the frequency of each query was the the most common activity they would undertake (4 out of 5 responses), reasoning that if numerous people search for the same type of service, that is a sign such a service should be launched, if it does not exist. PM2 and PM4 mentioned that several such datasets from different moments in time could be used to predict future requests, based on the evolution history. PM3 and PM4 also added that it would be interesting to analyze those queries that are similar to what their companies already offer, to identify what kind of changes could be implemented to enhance their services and satisfy consumers. The most advanced approach was suggested by PM5, who mentioned she would use the tf-idf (term frequency-inverse document frequency) method to determine how important a query is in a set of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2466538, IEEE Transactions on Services Computing

queries (document) of a collection of sets of queries (corpus). This is similar to analyzing individual query frequency, since the tf-idf is proportional to the frequency of each query, but it is offset by the frequency of the query in the corpus. When asked whether they utilized such datasets at all, three PMs answered that their recommender systems are the only ones making use of such data, to suggest services similar to the ones viewed by the user.

Therefore, although the five PMs interviewed are too few for achieving statistical significance, they confirmed our hypothesis that there are no elicitation methods in use that take the same type of input as our method, for the purpose of requirements acquisition. Moreover, our tool implementing the approach supports automating the activities suggested by the PMs, and goes beyond these. On the one hand, these findings encourage us to continue our efforts invested in elaborating the presented approach. On the other hand, they certify that finding a ground truth for an external evaluation is virtually impossible. Consequenty, we performed the evaluation described below.

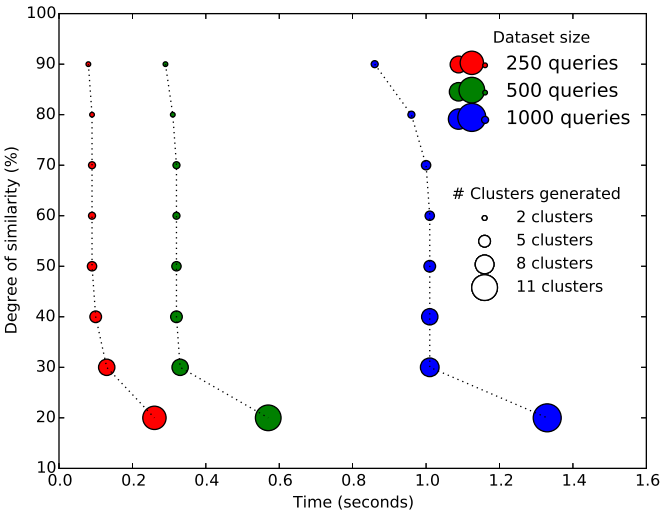

Figure 3. Time performance of the similarity classifier and number of clusters generated.

3.4 Experimental Setup

We used three distinct datasets as our simulation data, representing advanced search queries for data storage cloud services. They contain 250 queries[1], 500 queries[2] and 1000 queries[3], respectively. Since obtaining large amounts of real-world search information is particularly difficult due to sensitivity and privacy, we generated the needed datasets ourselves.

However, we used an available small dataset containing real-world data [18] coming from one of the companies interviewed [17] as a starting point. Moreover, we generated the queries such that they define data storage cloud services with ten features, as described in Table 1. For this, we followed the constraints imposed by the individual features: some are represented on a nominal scale, whereas others are represented on a ratio scale. Therefore, some of the features are defined by binary values, such as the "mobile support", and others are defined by fuzzy values in the range [0,1], such as "reliability". To test the performance of the similarity classifier, we set a degree of similarity between 20 and 90%.

All the experiments presented in the following subsections were run on a 4 GB 1333 MHz DDR3, 1.8 GHz Intel Core i7.

3.5 Automation and Time-Efficiency

We saw that existing requirements elicitation methods are challenged in the cloud domain by the time factor. Moreover, the need to involve a large number of stakeholders simultaneously in the same geographical spot is another issue posed by the cloud settings, which could be solved by means of automation. To show how our approach addresses these challenges, we firstly analyze the time efficiency (R2) of our algorithms and then describe the automation capabilities (R3).

1. http://www.ifi.uzh.ch/rerg/people/todoran/Dataset250.pdf
2. http://www.ifi.uzh.ch/rerg/people/todoran/Dataset500.pdf
3. http://www.ifi.uzh.ch/rerg/people/todoran/Dataset1000.pdf

3.5.1 *Time Efficiency (R2)*

In contrast to the existing elicitation techniques, our approach has a passive character, i.e. consumers are not directly and consciously involved in the requirements elicitation process, since the requirements for new services are inferred based on their searches. This way, virtually no time is dedicated specifically to the elicitation process, but rather to the data analysis. Consequently, we will calculate the data processing times.

Similarity Classifier

We firstly analyze the behavior of the similarity classifier (Algorithm 3). As described in Section 2.3.3, the degree of similarity (sim) is given by the cloud provider representative using the StakeCloud prototype [24] that implements our approach. Then, the bundles of similar queries are automatically generated. We measured the time needed to generate clusters for degrees of similarity between 20 and 90%, with an increment of 10, for the three datasets of 250, 500 and 1000 queries, respectively. We excluded the extreme margins of the range [0,100], i.e. [0,20) and (90,100], because values converging to 0 will not generate any clusters and will simply maintain the original list of queries, and values converging to 90 will likely generate only one large cluster consisting of all the queries in the dataset. Moreover, we measured how many clusters are generated for each degree of similarity, for every dataset. The results obtained are shown in Figure 3.

Generally, the time needed to process the data and generate queries bundles grows linearly with the number of queries in the dataset: e.g., for $sim = 20$, generating clusters for a dataset of 250 queries (shown in red in Figure 3) will take 0.26 seconds, while for a dataset of 500 queries (green) it will take 0.57 seconds, and for 1000 queries (blue) 1.33 seconds. This observation also holds for the other values of the degree of similarity. The time values we obtained are in the same range as those of well-known clustering methods, such as k-means [25], for datasets of comparable sizes.

As expected, the time required to generate the clusters is directly proportional to the number of clusters generated,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2466538, IEEE Transactions on Services Computing

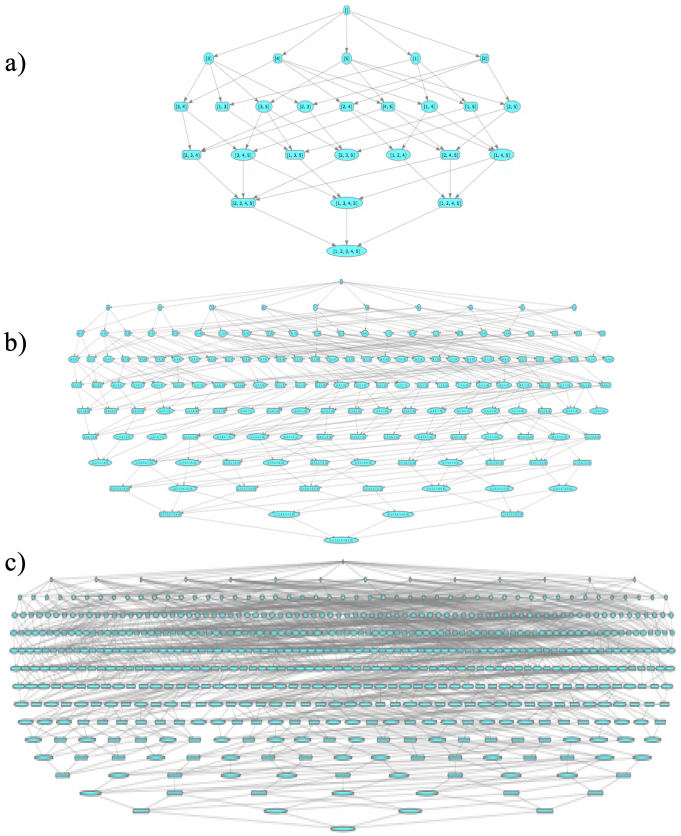

Figure 4. Fuzzy Galois lattices for: a) 5 queries, b) 10 queries, and c) 15 queries.

and inversely proportional to the value of sim. In Figure 3, the bigger the size of the circle representing the cluster, the larger the number of clusters generated for that particular degree of similarity, as exemplified in the legend. For instance, for the dataset with 500 queries (green), $sim = 90$ will lead to 2 clusters, whereas $sim = 20$ will lead to 11 clusters. It can be noted that the value of $sim = 20$ causes comparatively longer needed time periods compared to the other values. This is due to the fact that our algorithm always compares the distances between the supremum and each query to the breaking point calculated (line 15, in Algorithm 3), and the lower the value of sim, the more such computations needed. Nevertheless, the times are still manageable on a regular machine, being in the range of seconds for thousands of queries.

According to our tests so far, visualizing Galois lattices generated directly from datasets with less than 15 queries is still possible, without any need for clustering, unless specifically desired so. The tool allows zooming and dragging, thus making it easy for the user to navigate in the models even when the graphs are rather complex, as in Figure 4 b) and c). For datasets with more than 15 queries, the user can choose to automatically compute bundles and then generate the lattice. This way, (s)he can adjust the degree of similarity until the model is easy to visualize. When some of the represented nodes are not original queries from the dataset, but bundles representatives, a mouseover feature allows the user to see what composing queries each cluster consists of, and their frequencies.

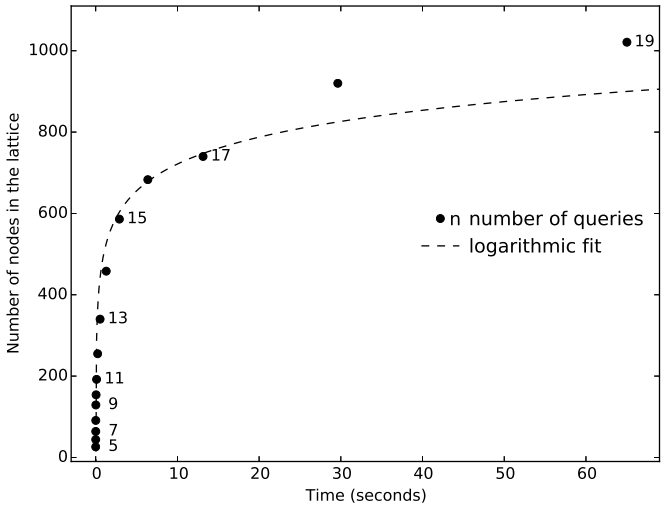

Figure 5. Time needed to generate the lattice nodes.

Lattice Generation

We computed the time needed to generate fuzzy Galois lattices for various datasets, containing between 5 and 19 queries. We tested our Algorithm 2 with datasets of these sizes, since these are likely to lead to models that are still easy to visualize and analyze, as explained above. For this reason, we generally recommend bundling similar queries initially, if the input dataset contains more than 15 queries, such that an overall view of the data is first generated. Then, the user can explore individual bundles and regenerate the model as needed. This also ensures a rapid lattice generation, thus fast data processing: e.g., building the model for 10 distinct queries takes 0.05 seconds, generating a total of 154 nodes to be represented in the lattice.

The graph in Figure 5 shows the time needed to generate the corresponding lattice nodes for the datasets used. The labels displayed next to the points represent the number of queries processed. Datasets of 5 to 13 queries produce lattices almost instantly, in less than 1 second. Then, the time needed increases with an exponential tendency. The logarithmic fit curve for the generated points is represented as a dotted line and has the following definition:

$$y(x) = 95.35 * log(29.52 * x) + 179.16$$

Whereas the method is indeed based on combinations of initial queries, several reductions are applied. Firstly, the frequency of each query is calculated, leading to representing only unique queries. In cases when frequency values higher than 1 are detected, these are labeled and propagated in the new classes of services generated. Secondly, when the vector of fuzzy values representing a query is the minimum of another vector, numerous nodes are generated, that are identical; again, only unique new classes of services are included in the lattice, which is another reduction opportunity scenario. Thirdly, queries in real world datasets do not tend to be thoroughly different, and the more similar they are, the fewer the lattice nodes. Therefore, despite its supposedly exponential character, our approach performs better than 2^n. The combinations initially computed by Algorithm 2 only concern the labels of queries and the actual calculations

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2466538, IEEE Transactions on Services Computing

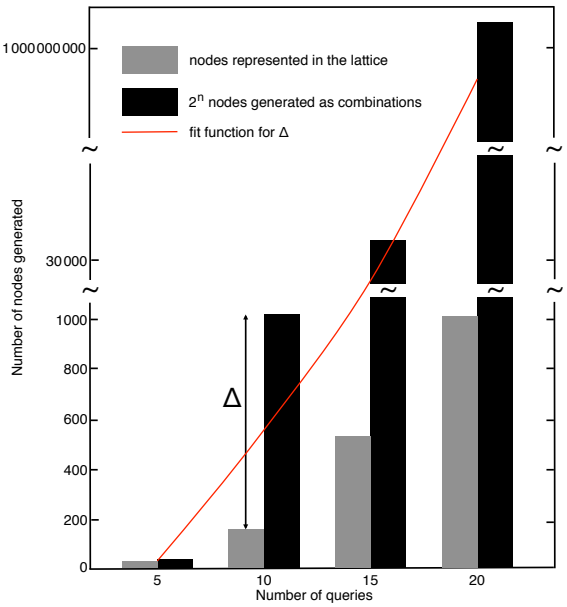

Figure 6. Lattice nodes vs. combinations.

are only performed on the nodes that qualify to be part of the lattice.

The reductions applied lead to high delta (Δ) values between the number of combinations generated mathematically and the actual number of nodes needed to build the lattice. This is shown in Figure 6, that presents the discrepancy between the number of lattice nodes (grey) and the combinations (black), for 5, 10, 15 and 20 queries, respectively. For instance, for a dataset of 15 queries, $\Delta = 1047464$. The exponential fit curve for the middle points of Δ values is shown in red, and has the definition:

$$y(x) = 0.54 * e^{0.69*x}$$

The exponential character of this fit curve demonstrates that while the time needed to generate the lattices grows fast, the approach performs better than standard exponential, not needing all the mathematical combinations. When the input datasets contain less than 50 unique queries, the computing time remains in the range of hours, e.g., for 25 queries, without any pre-processing and bundling, the needed time is 3.85h. Nevertheless, as soon as bundling is applied, the time can drop drastically, depending on the value of the degree of similarity. This efficiency issue is known in the field of concept lattices, and various solutions have been investigated to mitigate it. For instance, Kumar and Srinivas [26] apply k-means clustering for lattice reduction, defining a number of desired clusters k. However, they only focus on the clusters resulted in their analysis, not mentioning anything about the performance gain. In our context, we can conclude that the time required for the data analysis is still much shorter than that of any other existing requirements elicitation techniques, even when several hours of computation are needed. Moreover, the results reported are obtained on our rather limited desktop machine. Algorithm 2 is principally parallelizable, and running it on a multi-core cluster should yield better results, decreasing the computation time proportionally to the number of cores used. However, this is subject to future work.

3.5.2 Automated Elicitation (R3)

As far as automation is concerned, our Galois lattices approach for cloud services is different from all the existing requirements elicitation techniques in that data collection is exclusively automated. The needs are gathered from (potential) consumers in a passive and unobtrusive way. Moreover, our technique is tool-supported [24], such that most of the data analysis is automated. As shown in Figure 2, the StakeCloud Tool performs numerous activities, such as computing bundles of similar queries. Whereas the new classes of services are automatically generated, while performing the analysis in A10, providers can perform a manual what-if investigation to dynamically simulate what happens when only one or a few features are varied, how these impact the general clustering, or zoom in specific parts of the lattice, to analyze the best ideas for new services.

For instance, Figure 7 shows a sample scenario in our tool. The top right panel contains the input dataset loaded, consisting of 20 distinct queries. The corresponding Galois lattice is displayed in the main panel, where the topmost element is the supremum for the entire lattice, and the first level in the hierarchy is composed of inidvidual queries (circles, e.g., *(6)*) and bundles (rounded rectangles, e.g., *(2)*) generated based on a degree of similarity of 75%. The remaining nodes are classes of services generated automatically, as infima of the elements in the first hierarchy level.

Upon selecting three latice nodes displayed in gray (the bundle *(2)* and the individual queries *(1)* and *(4)*), the user can immediately visualize the corresponding supremum (*()*, in green) and infima elements (red). The numerical fuzzy values of these are also displayed in the right central panel. As explained in Section 2.2 A7, the infima elements are the main candidates to evaluate when the provider is interested in satisfying consumers with searches such as *(1)*, *(2)* and *(4)*. For this, the user can select from five different criteria of analysis from the drop-down menu in the bottom right panel, e.g., analyze to what extent individual features are accomplished by selected nodes or their suprema/infima. Assuming the cloud provider is interested in satisfying as many as possible features fully for the selected queries, he selects the suprema/infima full satisfaction analysis.The tool automatically generates the graph displayed in the bottom right corner of the tool window, showing the results. The graph allows multiple plotting options, the scale can be changed and zooming capabilities are also embedded.

As a next step, the user can decide to choose other analysis criteria, look into the individual queries composing the bundle *(2)*, or regenerate the lattice by changing the degree of similarity (as shown by the loops in Figure 2). Moreover, there is the option of adding new queries to the dataset. Lastly, the approach allows the user to model his/her existing offerings as vectors of fuzzy values, and then to search if these are among the lattice nodes. If they are found, they are highlighted, which helps cloud providers to see what queries they can satisfy with their current services, or how they could mildly enhance them to satisfy larger populations. In case their current services are not found

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2466538, IEEE Transactions on Services Computing

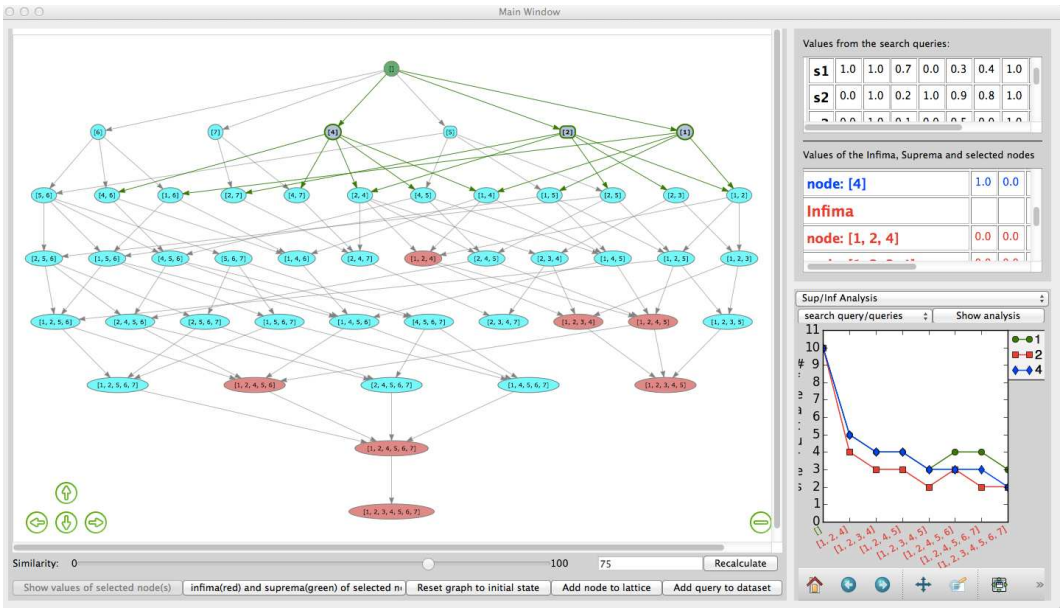

Figure 7. Tool screenshot - sample scenario.

among the nodes, a new lattice including them is generated, showing their relations to the other lattice elements.

Therefore, through the tool support provided, we propose our approach as a requirements elicitation technique that allows the automation of requirements acquision to a large extent, and puts at cloud providers' disposal means for analyzing the automatically generated visualizations.

3.6 Consumers' Heterogeneity, Geographical Distribution and Volatile Requirements

In this subsection, we show how our approach addresses R1, R4 and R5, by enabling the elicitation of requirements from globally distributed audiences, whose needs are volatile.

3.6.1 Wide and Heterogeneous Audiences (R1)

The advanced searches needed by our fuzzy Galois lattices technique are always conducted on marketplaces websites. Therefore, our approach allows any number of consumers from virtually anywhere to input their needs for services in a completely asynchronous way. Moreover, according to the interviewed cloud providers [17], performing advanced searches for services is among the most frequent methods used by both individual consumers and businesses to select cloud solutions suitable for their needs.

When generating the datasets used for experimentation, we took into account the heterogeneity aspect. Therefore, the queries used are highly heterogeneous, while they still follow the constraints posed by the features in Table 1. As a metric for heterogeneity, we use the standard deviation:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

In our case, N is the number of queries, x_i, $i = 1, N$ represent the individual queries, and μ is the mean of the queries in the dataset. We used the standard deviation for measuring the amount of dispersion from the average for the queries in our three datasets, and obtained the following results. For the dataset composed of 250 distinct queries, $\sigma_{250} \simeq 0.324$, for the 500 queries, $\sigma_{500} \simeq 0.322$, and for the 1000 queries, $\sigma_{1000} \simeq 0.319$. All these values round to the value of 0.32, indicating that all the queries used in our tests are well spread, therefore heterogeneous.

3.6.2 Remote Application (R4)

Our approach is search-based, which means that it can be applied for any consumers, located anywhere, including those who are not physically reachable.

According to Use Case 3.9 defined for cloud computing by NIST (American National Institute of Standards and Technology)[4], "a cloud-user makes a structured capability or capacity or price request to one or several cloud-providers and receives a structured response that can be used as input to drive service decisions". This use case describes exactly the paradigm our approach is built on: the remote request for cloud service capabilities, upon which consumers get matching results from the marketplace. Such remote requests are the queries used as input, enabling the remote application of our approach. Moreover, remote analysis of the output of our method is also possible.

For instance, in the example shown in Figure 7, the automatically generated bottom right graph represents the number of features from queries *(1)* (green) and *(4)* (blue) and bundle *(2)* (red) that are fully satisfied by their supremum and infima. The information displayed in such graphs is available to the cloud provider company without the need to send its consultants overseas. For example, the cloud provider representative can observe that classes of services like *(1,2,4,5,6)* and *(1,2,4,5,6,7)* would both satisfy fully 4 features for query *(1)* and 3 features for query *(4)*. However, *(1,2,4,5,6,7)* satisfies one feature less than *(1,2,4,5,6)* for bundle *(2)*. Nevertheless, it has the advantage of satisfying also query *(7)*, which belongs to the composing set. In this

4. $http://www.nist.gov/itl/cloud/3\_9.cfm$ Accessed: June 2015.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2466538, IEEE Transactions on Services Computing

case, it is worth studying what is more valuable for the cloud provider: satisfying bundle *(2)* to a larger extent, or query *(7)*. The analysis can continue until the provider has obtained enough information to make a decision what service(s) are worth launching, e.g., by unbundling query *(2)* to analyze the queries it consists of, generating the graphs that visualize to what extent query *(7)* can be satisfied by the given infima, generating the graphs that show also the extent to which the features of the selected nodes are partially satisfied, etc. For another analysis example, please refer to [18]. This analysis is performed exclusively remotely, based on the input dataset.

Given its unobtrusive character, this technique is also suitable when consumers are not able to describe their requirements easily in an interview or a workshop, and can thus be used to complement other elicitation methods.

3.6.3 Volatile Requirements (R5)

Volatile requirements, i.e. requirements that change while being elicited, analyzed, validated and/or implemented, represent a recognized challenge for requirements elicitation methods. In contrast to the existing techniques, where extensive time is allocated to first gathering requirements which are then analyzed, our approach enables a continuous elicitation process. Cloud consumers' data is collected continuously as advanced search queries on marketplaces, and can be instantly fed as input to our tool-supported approach. This way, volatile requirements can be monitored in an uninterrupted fashion. Moreover, due to the remote character of our method, this is done at virtually no additional cost. This feature is particularly fitting with the agile character of most cloud provider companies, which use fast development cycles and have rather short times to market.

Furthermore, our approach can be used for trend monitoring. For instance, it can calculate the mean for selected features of a series of datasets over a period of time. If it is detected that the mean value shifts steadily over that period of time, this represents a hint that the feature might follow that trend also in the future. For example, if the mean for the feature "storage" increases by 0.5GB every quarter for two years, this may indicate that a growth should be expected also in the following year(s). Conversely, if the frequency of a binary feature such as "AES encryption" decreases over a given series of datasets, this may be an indication that this features may be replaced by another, or consumers simply do not want it any more.

3.7 Threats to Validity

As far as construct validity is concerned, we tried to avoid evaluation apprehension by ensuring the product managers contacted that all the information is anonymized and used exclusively for research purposes. We also mitigated hypothesis guessing by not giving them any details about our approach. Internal validity is threatened by the fact that we generated the datasets ourselves. However, we constructed them respecting the constraints of the features encoded and starting from a real world dataset. This way, we reduced the possible causal relationship between treatment and outcome. The fact that the datasets are self-generated concerns the external validity, i.e. generalizing results to industrial practice in particular. However, we attempted to build representative queries, as described above.

4 Related Work

From the early 2000s, researchers observed that requirements engineering also needs to consider distributed [27] and asynchronous settings [28], and this currently extends to the cloud context. However, due to its collaboration-intensive and time-consuming nature, requirements elicitation becomes difficult in the cloud [27], [29].

As far as *dedicated cloud requirements elicitation methods* are concerned, there has been some advancement during the recent years. For instance, frameworks focusing on the supply-demand relation have been designed [30], Sun et al. developed a hybrid fuzzy framework for helping cloud consumers to select services that match their needs when their requests are uncertain [31], and management systems for requirements ensuring QoS have been developed [32]. Moreover, researchers looked into methods for eliciting particular types of requirements, e.g., legal [33] or security [34]. Still, these are only niche recommendations and no comprehensive clear solution exists, addressing the cloud-specific requirements elicitation challenges.

As far as *distributed requirements elicitation* is concerned, Lloyd et al. conducted a study [15] on the effectiveness of elicitation techniques in distributed requirements engineering, concluding that synchronous elicitation approaches are generally more effective than asynchronous ones. Lim et al. [35] present ideas on asynchronous and distributed stakeholder identification, assuming that key stakeholders are known, and further users can be identified based on domain knowledge. However, such approaches do not easily extend to the cloud context, since the audience for services is most often unknown and globally distributed.

Tuunanen [36] addresses the problem of reaching and involving wide audience end-users, or users who are not within organizational reach. He argues that traditional techniques do not provide adequate solutions and presents methods which could potentially fill this gap (e.g., Easy-WinWin). However, none of these methods has been successfully used on a large scale for distributed elicitation so far. Moreover, research on EasyWinWin by Kukreja et al. [37] promises to provide support for distributed settings, but only focuses on stakeholders within organizational reach.

Another challenge of requirements elicitation in the cloud is the continuous change of consumer needs [38]. Consequently, various wiki approaches have been implemented, to provide a time-efficient possibility for updating and eliciting requirements. For instance, Decker et al. developed a wiki-based solution that enables stakeholders' participation in RE [39], Solis and Ali's spatial hypertext wiki focuses on distributed teams [40], whereas Liang et al. [41] and Lohmann et al. [42] exploit semantic annotation wikis. However, wiki-based methods generally assume that stakeholders are at least identifiable, which is not the case in a cloud context.

Studies from the field of web-information systems by Yang and Tang [43] reveal that the elicitation needs regarding Internet-based systems are also rather different from

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2015.2466538, IEEE Transactions on Services Computing

those of traditional systems (e.g., due to higher user diversity). Moreover, most existing requirements elicitation methods can only deal with a limited number of stakeholders [38]. The number of potential cloud service consumers may often go beyond what traditional methods can handle [34], and no real solutions addressing this elicitation issue have been developed so far. Market-driven techniques [44], which are usually employed when it is impossible to consider individual consumers, prove to be rather limited in the cloud, due to the lack of specific localized markets.

Work in data mining, machine learning and particularly *recommender systems* [5] also addresses the problem of extracting value from search data. For example, search-based and collaborative techniques can make personalized online product recommendations [45], and user feedback has been used to rank various products [46]. Throughout the recent years, recommender systems [23] (e.g., probability-based collaborative filtering [47]) and clustering data mining methods [48] have been heavily used for marketing purposes, to suggest similar products in e-commerce systems, or to segment populations. However, to our knowledge, such techniques have never been adapted or utilized for requirements elicitation in the cloud.

Furthermore, data analysis methods that model consumers' needs have not been explored for the purpose of cloud requirements elicitation so far. Whereas there is an extensive body of research in the field of Galois concept lattices, most researchers exclusively focus on the mathematical implications of such graphs, and do not apply them in a practical context. There are only a few examples of attempts where lattices are used to identify objects or concepts in given datasets [49], [50], or applied to browsing retrieval [51]. Most existing lattice-related work focuses on clustering opportunities with Galois lattices from a purely theoretical perspective [52].

To summarize, the existing elicitation techniques, even when adapted, are mostly unsuitable for the cloud, and can support cloud service providers only to a limited extent in their requirements elicitation processes. Moreover, the existing data mining and lattice approaches have not been applied in the requirements acquisition context so far, to our knowledge, leaving the issue of dedicated cloud requirements elicitation techniques unsolved.

5 Conclusion and Future Work

In this paper, we proposed a new approach to requirements elicitation for cloud services, that builds fuzzy Galois lattices based on consumers' advanced search queries. We evaluated it against five main cloud providers' requirements and showed how it addresses them: it can gather needs from wide and heterogeneous audiences whose requirements are volatile, automates data analysis and can be applied remotely, taking less time than traditional elicitation methods. Our approach is best-suited for the early elicitation phase and for monitoring market trends. It can be succeeded by more in-depth requirements elicitation with complementary methods such as prototyping and large-scale online experiments. Although our approach was natively built for cloud contexts, we foresee that it can be applied successfully also in other domains that exhibit similar properties and where queries can be collected in a similar fashion, e.g., in traditional web-service and service-oriented systems. However, we have not investigated this usage scenario so far, but it is subject to future work.

A limitation of our approach is that it assumes consumers provide values for all the features specified as advanced search criteria. Currently, we ignore the queries with values of zero for all features (outliers) and allocate default values when no values are assigned. We are now working on implementing the maximum likelihood estimation, which uses the available data to compute maximum likelihood estimates. Moreover, we plan to improve the time performance and method scalability by using MapReduce to compute the minima values in Algorithm 2. In addition, we are currently working on extending the approach to work when particular features have weights denoting their importance. Naturally, adoption issues related to users' data privacy may occur when our method is deployed in practice and some time may be needed until cloud providers get used to the workflow. Therefore, we plan to apply our elicitation method in real world settings, with cloud providers and their datasets, to discover any potential issues and demonstrate its actual impact and practical use.

Acknowledgments

The authors would like to thank Matthias Scherrer for his contribution. This work is supported by the Swiss National Science Foundation (SNSF), grant no. 200021_150142.

References

[1] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the cloud? An architectural map of the cloud landscape," in *ICSE Workshop on Software Engineering Challenges of Cloud Computing*. IEEE Com Soc, 2009, pp. 23–31.

[2] K. S. Candan, W.-S. Li, T. Phan, and M. Zhou, "At the frontiers of information and software as services," in *New Frontiers in Information and Software as Services*. Springer, 2011, pp. 283–300.

[3] B. Schwartz, "The paradox of choice: Why less is more," *New York: Ecco*, 2004.

[4] J. Kang and K. M. Sim, "Cloudle: a multi-criteria cloud service search engine," in *IEEE Asia-Pacific Services Computing Conference (APSCC)*. IEEE, 2010, pp. 339–346.

[5] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Trans on Knowledge and Data Eng*, vol. 17, no. 6, pp. 734–749, 2005.

[6] I. Todoran, "Stakecloud: Stakeholder requirements communication and resource identification in the cloud," in *20th IEEE Intl Requirements Eng Conf (RE'12)*. IEEE, 2012, pp. 353–356.

[7] Intel Cloud Finder. Accessed: June 2015. [Online]. Available: http://www.intelcloudfinder.com/

[8] A. Khajeh-Hosseini, D. Greenwood, J. W. Smith, and I. Sommerville, "The cloud adoption toolkit: supporting cloud adoption decisions in the enterprise," *Software: Practice and Experience*, vol. 42, no. 4, pp. 447–465, 2012.

[9] S. Zardari, R. Bahsoon, and A. Ekart, "Cloud adoption: Prioritizing obstacles and obstacles resolution tactics using ahp," in *29th ACM Symp On Applied Computing, Req Eng Track*, 2014.

[10] S. Zardari and R. Bahsoon, "Cloud adoption: a goal-oriented requirements engineering approach," in *2nd Intl Workshop on Software Eng for Cloud Computing*. ACM, 2011, pp. 29–35.

[11] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," in *Conf on the Future of Software Eng*. ACM, 2000, pp. 35–46.

[12] I. Sommerville and G. Kotonya, *Requirements engineering: processes and techniques*. John Wiley & Sons, Inc., 1998.

[13] A. Van Lamsweerde, "Requirements engineering: from system goals to uml models to software specifications," 2009.

[14] B. Nuseibeh, S. Easterbrook, and A. Russo, "Leveraging inconsistency in software development," *Computer*, vol. 33, no. 4, pp. 24–29, 2000.

[15] W. J. Lloyd, M. B. Rosson, and J. D. Arthur, "Effectiveness of elicitation techniques in distributed requirements engineering," in *IEEE Joint Intl Conf on Requirements Eng*. IEEE, 2002, pp. 311–318.

[16] T. Tsumaki and T. Tamai, "Framework for matching requirements elicitation techniques to project characteristics," *Software Process: Improvement and Practice*, vol. 11, no. 5, pp. 505–519, 2006.

[17] I. Todoran, N. Seyff, and M. Glinz, "How cloud providers elicit consumer requirements: An exploratory study of nineteen companies," in *21st IEEE Intl Requirements Eng Conf (RE'13)*, 2013, pp. 105–114.

[18] I. Todoran and M. Glinz, "Quest for requirements: Scrutinizing advanced search queries for cloud services with fuzzy Galois lattices," in *IEEE 10th World Congress on Services (SERVICES)*. IEEE, 2014, pp. 234–241.

[19] K. Denecke, M. Erné, and S. L. Wismath, *Galois Connections and Applications*. Kluwer Academic Publishers, 2004.

[20] M. Erné, J. Koslowski, A. Melton, and G. E. Strecker, "A primer on Galois connections," *Annals of the New York Academy of Sciences*, vol. 704, no. 1, pp. 103–125, 1993.

[21] R. Belohlavek, "Fuzzy Galois connections," *Mathematical Logic Quarterly*, vol. 45, no. 4, pp. 497–504, 1999.

[22] B. Ganter, R. Wille, and C. Franzke, *Formal concept analysis: mathematical foundations*. Springer-Verlag New York, 1997.

[23] P. Resnick and H. R. Varian, "Recommender systems," *Commun ACM*, vol. 40, no. 3, pp. 56–58, 1997.

[24] I. Todoran Koitz and M. Glinz, "Stakecloud Tool: From cloud consumers' search queries to new service requirements," *Accepted for publication in the 23rd IEEE Intl Requirements Eng Conf (RE'15) - to appear*, 2015.

[25] S. Chakraborty, N. Nagwani, and L. Dey, "Performance comparison of incremental k-means and incremental DBSCAN algorithms," *International Journal of Computer Applications*, vol. 27, no. 11, pp. 14–18, 2011.

[26] C. Aswani Kumar and S. Srinivas, "Concept lattice reduction using fuzzy k-means clustering," *Expert systems with applications*, vol. 37, no. 3, pp. 2696–2704, 2010.

[27] D. E. Damian and D. Zowghi, "RE challenges in multi-site software development organisations," *Requirements Eng*, vol. 8, no. 3, pp. 149–160, 2003.

[28] P. Grünbacher and N. Seyff, "Requirements negotiation," in *Engineering and Managing Software Requirements*. Springer, 2005, pp. 143–162.

[29] A. Daugulis, "Time aspects in requirements engineering: Or 'every cloud has a silver lining'," *Requirements Eng*, vol. 5, no. 3, pp. 137–143, 2000.

[30] L. Liu, *Software Reuse in the Emerging Cloud Computing Era: Goal-Based Requirements Elicitation for Service Reuse in Cloud Computing*. IGI Global, 2012.

[31] L. Sun, H. Dong, F. K. Hussain, O. K. Hussain, J. Ma, and Y. Zhang, "A hybrid fuzzy framework for cloud service selection," in *IEEE Intl Conf on Web Services (ICWS)*. IEEE, 2014, pp. 313–320.

[32] D. Villegas and S. M. Sadjadi, "Mapping non-functional requirements to cloud applications," in *SEKE*, 2011, pp. 527–532.

[33] K. Beckers, S. Fassbender, and H. Schmidt, "An integrated method for pattern-based elicitation of legal requirements applied to a cloud computing example," in *7th Intl Conf on Availability, Reliability and Security (ARES)*, 2012, pp. 463–472.

[34] K. Beckers, M. Heisel, I. Cote, L. Goeke, and S. Guler, "Structured pattern-based security requirements elicitation for clouds," in *8th Intl Conf on Availability, Reliability and Security*, 2013, pp. 465–474.

[35] S. L. Lim, D. Quercia, and A. Finkelstein, "Stakenet: using social networks to analyse the stakeholders of large-scale software projects," in *32nd ACM/IEEE Intl Conf on Software Eng, vol. 1*. ACM, 2010, pp. 295–304.

[36] T. Tuunanen, "A new perspective on requirements elicitation methods," *Journal of Inf Tech Theory & Application*, vol. 5, no. 3, 2003.

[37] N. Kukreja and B. Boehm, "Process implications of social networking-based requirements negotiation tools," in *Intl Conf on Software and Sys Process (ICSSP)*. IEEE, 2012, pp. 68–72.

[38] J. D. Herbsleb, "Global software engineering: The future of socio-technical coordination," in *Future of Software Eng*. IEEE Computer Society, 2007, pp. 188–198.

[39] B. Decker, E. Ras, J. Rech, P. Jaubert, and M. Rieth, "Wiki-based stakeholder participation in requirements engineering," *IEEE Software*, vol. 24, no. 2, pp. 28–35, 2007.

[40] C. Solis and N. Ali, "Distributed requirements elicitation using a spatial hypertext wiki," in *5th IEEE Intl Conf on Global Software Eng (ICGSE)*, 2010, pp. 237–246.

[41] P. Liang, P. Avgeriou, and V. Clerc, "Requirements reasoning for distributed requirements analysis using semantic wiki," in *4th IEEE Intl Conf on Global Software Eng (ICGSE)*, 2009, pp. 388–393.

[42] S. Lohmann, T. Riechert, S. Auer, and J. Ziegler, "Collaborative development of knowledge bases in distributed requirements elicitation." in *Software Eng (Workshops)*, vol. 122, 2008, pp. 22–28.

[43] H.-L. Yang and J.-H. Tang, "A three-stage model of requirements elicitation for web-based information systems," *Ind Mgmt & Data Sys*, vol. 103, no. 6, pp. 398–409, 2003.

[44] L. Karlsson, Å. Dahlstedt, J. N. och Dag, B. Regnell, and A. Persson, "Challenges in market-driven requirements engineering - an industrial interview study," in *8th Intl Workshop on Requirements Eng: Foundation for Software Quality (REFSQ'02)*, 2003, pp. 101–112.

[45] N. Abdullah, Y. Xu, and S. Geva, "Integrating collaborative filtering and search-based techniques for personalized online product recommendation," in *11th IEEE Intl Conf on Data Mining Workshops (ICDMW'11)*, 2011, pp. 711–718.

[46] T. Q. Lee, Y. Park, and Y.-T. Park, "A time-based approach to effective recommender systems using implicit feedback," *Expert Systems with Applications*, vol. 34, no. 4, pp. 3055–3062, 2008.

[47] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Trans on Inf Sys (TOIS)*, vol. 22, no. 1, pp. 5–53, 2004.

[48] P. Berkhin, "A survey of clustering data mining techniques," in *Grouping Multidimensional Data*, J. Kogan, C. Nicholas, and M. Teboulle, Eds. Springer Berlin Heidelberg, 2006, pp. 25–71.

[49] A. Van Deursen and T. Kuipers, "Identifying objects using cluster and concept analysis," in *21st Intl Conf on Software engineering*. ACM, 1999, pp. 246–255.

[50] R. Godin, R. Missaoui, and H. Alaoui, "Incremental concept formation algorithms based on Galois (concept) lattices," *Computational intelligence*, vol. 11, no. 2, pp. 246–267, 1995.

[51] C. Carpineto and G. Romano, "A lattice conceptual clustering system and its application to browsing retrieval," *Machine learning*, vol. 24, no. 2, pp. 95–122, 1996.

[52] N. Pernelle, M.-C. Rousset, H. Soldano, and V. Ventos, "Zoom: a nested Galois lattices-based system for conceptual clustering," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 14, no. 2-3, pp. 157–187, 2002.

Irina Todoran Koitz received her BSc degree in computer science from the Polytechnic University of Bucharest, Romania, and the MSc degree with honors in service design and engineering from Aalto University (Helsinki University of Technology), Finland. She is currently working on her PhD in computer science at the University of Zurich, under the supervision of Martin Glinz. Her interests include cloud requirements communication, data analysis and empirical studies.

Martin Glinz is a full professor and head of the Department of Informatics at the University of Zurich. His interests include requirements and software engineering - in particular modeling, validation, and quality - and software engineering education. He received his Dr. rer. nat. in Computer Science from RWTH Aachen University. Before joining the University of Zurich, he worked in industry for ten years.